A taxonomy of software types to facilitate search and evidence-based software engineering

Conference Paper · January 2008

DOI: 10.1145/1463788.1463807 · Source: DBLP

2 authors:

Andrew Forward
University of Ottawa
28 PUBLICATIONS **1,009** CITATIONS

SEE PROFILE

Timothy Lethbridge
University of Ottawa
208 PUBLICATIONS **5,437** CITATIONS

SEE PROFILE

A Taxonomy of Software Types to Facilitate Search and Evidence-Based Software Engineering

Andrew Forward
Timothy C. Lethbridge

University of Ottawa
800 King Edward
Ottawa, Ontario, Canada K1N 6N5
Telephone: + 1 613 562 5800 x 6685
Email: {aforward,tcl}@site.uottawa.ca

Abstract

Empirical software research could be improved if there was a systematic way to identify the types of software for which empirical evidence applies. This is because results are unlikely to be globally applicable, but are more likely to apply only in certain contexts such as the type of software on which the evidence has been tested. We present a software taxonomy that should help researchers to apply their research systematically to particular types of software. The taxonomy was generated using existing partial taxonomies and input from survey participants. If a taxonomy such as ours gains acceptance, it will facilitate comparison and appropriate application of research. In the paper, we present the benefits of such a taxonomy, the process we used to develop it, and the taxonomy itself.

1 Introduction

In this paper, we present a taxonomy of types of software. We would eventually like to see a descendent of this evolve into a standard that would be referenced by both researchers and practitioners.

This taxonomy can be used to help categorize empirical studies and other evidence obtained by software engineering researchers, so that the applicability of the evidence can be more systematically understood. The taxonomy can also be used in the practical application of research results. For example, research findings for a particular software category might be applicable in closely related categories within the taxonomy.

Software engineering practices, principles and pragmatics should not be adopted just because someone had proposed them or found them to work in certain contacts. Rather, they must be both evidence-based and contextual. In other words, there should be ideas behind which there is evidence, and furthermore the evidence should support the truth of the idea under a well-defined set of conditions.

One of the key conditions is the type of software being built. Clearly many types of software engineering activities and many principles will differ markedly whether one is developing real-time software or data processing software. However, these represent just two broad categories of software, there will also be important differences among many other kinds of software when it comes to choosing the appropriate software engineering techniques.

Ideally, then, any paper published containing a practical or empirical study should be able to say which classes of software it applies to. Unfor-

tunately, however, there is no published taxonomy or systematization of software types.

The notion of taxonomy, a structured vocabulary, is one of the oldest and most widely used modeling tools to organize information. Almost all newer conceptual models like attributes, tagging and axioms are all backed by a hierarchy [1], making developing a taxonomy a logical starting point to uncover software types.

A taxonomy is more than just a structure for categorizing familiar names. It represents a theory according to which those names are organized – defining how like and unlike names are distinguished [2]. The taxonomy presented in this paper is therefore a theory of software types.

Maintaining a software taxonomy provides the following benefits for software engineering research:

- **Improved software research.** A software taxonomy should be able to provide a context for empirical results in software engineering, as well as facilitate exploring the applicability of those results. The taxonomy would be used to suggest the types of software on which results could be tested. For example, a survey of software practitioners could be subdivided into participants that work with data-dominant versus computation-dominant software (which are two of the top-level categories in our taxonomy). From that subsampling the re-searcher might be able to demonstrate that a particular software process (or architecture, or framework, or testing technique, etc.) is indeed appropriate in both types of applications, or is in fact only applicable to one of these.

- **Artifacts more readily reusable.** Artifacts such as libraries, plugins, software patterns, and algorithms could be more easily reused if mapped to categories within a software taxonomy. The taxonomy could help identify candidate artifacts for reuse, as well as identify gaps – i.e. application types where more reusable artifacts are required.

- **Increased use of reference models** (sketches of the architecture) **and frameworks**. If reference models and frameworks were mapped to a taxonomy of software types, developers would have a better starting point when building new software in a particular domain. Similar to reusable artifacts above, the taxonomy will also identify which application types lack frameworks and reference models.

- **Appropriate education coverage.** With a software taxonomy, educators and curriculum designers would be able to build courses and programs that provide exposure to a broad range of application types.

1.1 Background

The computing field to date has focused primarily on application-independent approaches [3], or approaches that are specific to only a few categories such as real-time software. We are not questioning whether or not progress has been made by focusing on application-independent research, but that it is also important to consider application-domain-specific research that yields results for a particular class of problem.

Vessey's theory of Cognitive Fit [4] states that any methods, tools, or techniques in the software field should be specific to the type of application being addressed.

In 1992, Glass described the need for a more rigorous approach to define a software taxonomy and he outlined two application domain taxonomies [5].

In 1995, Glass provided a more in-depth analysis of existing taxonomies in the software field [6]. Glass argued that the classifications had not been developed with a consistent, and rigorous approach. There had been no formal development of application-focused taxonomies, no explicit consideration of classification criteria and no systematic attempt to build a classification method.

Although we found no all-encompassing taxonomy, we were able to identify 22 lists or partial lists of application types and the categories used by various software vendors and search sites. What follows is a look at well-known taxonomies from the identified list that include some form of software type categorization.

1.2 Existing Taxonomies

It is important to have a good understanding of existing taxonomies in the computer science field before creating a new one for software applications. The criteria for review include:

- Is the purpose and applicability of the taxonomy well-defined?
- Does the taxonomy appropriately cover the space we are interested in: software types (including both application program types, system software types, and software component types)?
- Is the taxonomy structured in a simple and systematic way, such as to minimize multiple inheritance and is it using well-defined criteria for dividing each category into subcategories?

One well-known taxonomy is the ACM computing taxonomy. This has undergone three official versions including 1964, 1991, and 1998. For the review, we will consider the most recent, 1998, edition.

The ACM taxonomy encompasses categories of computing endeavors and research, and is not designed to be a taxonomy of types of software. Nevertheless it does contain some parts that are lists of application domains. It is comprised of 1472 entries, or 1422 entries if you ignore 'miscellaneous', 405 entries if you consider only application-like categories and only 58 if you limit your analysis to two sections: J Computer Applications (50 entries) plus K.8 Personal Computing (8 entries). Although these latter two categories are dedicated categories for computer application types, there are over 350 additional categories scattered throughout the taxonomy that might be relevant, such as: word processing, spreadsheets, real-time and embedded systems, program editors, and software configuration management.

The Section J categories (only the first level) are shown below.

- J.0 GENERAL
- J.1 ADMINISTRATIVE DATA PROCESSING
- J.2 PHYSICAL SCIENCES AND ENGINEERING
- J.3 LIFE AND MEDICAL SCIENCES
- J.4 SOCIAL AND BEHAVIORAL SCIENCES
- J.5 ARTS AND HUMANITIES
- J.6 COMPUTER-AIDED ENGINEERING
- J.7 COMPUTERS IN OTHER SYSTEMS (C.3)

The ACM taxonomy's overall purpose is well-defined and well known – to provide a meaningful categorization of technology-related topics for research, review and reporting. From an application domain standpoint, it provides relatively poor coverage for certain branches of application types such as system software (e.g. operating systems), information display (e.g. maps / traveling) and consumer-oriented software (e.g. entertainment). The ACM taxonomy is a great starting point, and this research is heavily based on its recognized and well-used structure.

More recently, the open-source community has provided a rich playground for software to be built, shared, and improved. From a classification perspective, sites such as SourceForge and GoogleCode provide a good indicator of most types of software people are developing. Below we will look at Google's classification system.

Google approaches application domains slightly differently; and avoids using a typical hierarchical structure. Instead, Google relies mostly on non-hierarchical tagging of applications. The primary categories where application-appropriate tags were found are shown below:

- Computer Science
- Programming
- Software
    - Software Engineering
- Technology
    - Computational Engineering

For the needs of this research, Google's list is poorly defined, having no selection criteria. But despite this disorganization, it does provide excellent coverage. Google's directory has over 300 software-related entries and provides 103 entries relevant to software application domains. These application domains provide an excellent resource to ensure maximal coverage in the final result of our research.

Another approach to categorizing software could focus on the type of problem it solves. The literature on Problem Frames [7] [8], for example, suggests some important high-level categories.

The contribution of our research is to tie together and appropriately select from the taxonomies described above and other research already accomplished in this field. The need and justification for a software taxonomy has been formalized by Vessey and Glass [9]; a review of the existing

landscape to identify suitable candidates and / or properties of an application taxonomy has been done (albeit from 16 years ago) [5]; and, the necessary steps to complete this work have been iterated [6]. Our research builds on the work identified above and in particular it fills the gaps previously identified by Vessey, and Glass.

We used a systematic process to develop a software taxonomy to cover all types of software currently developed. The taxonomy should be useful to both researchers and practitioners who need to perform such tasks as cataloguing, filing or searching for applications. The taxonomy will also facilitate tagging of components and techniques according to the application types for which they are suitable.

2 Development Process

This section describes the process we followed to develop the taxonomy.

As is common with the development of software engineering standards like SWEBOK and SE2004, the process involved considerable input from both published material and experts, iterative review, and careful editing of the results (in this case by the co-authors) to improve cohesiveness.

We do not claim that the end-result is of standards-level quality, mainly because a wider consultation would be needed and a more open committee would need to be formed. Nevertheless, we believe the process we followed is capable of producing a credible *first cut* that can actually be put to use on an interim basis by researchers.

2.1 Steps in the Process

The taxonomy was created using the following multi-step process.

Step 1: Seed the process with existing taxonomies. First, we searched for existing taxonomies as described in Section 1.1. The research revealed no all-encompassing taxonomy so we set out to build our own. As a starting point, we used items in existing partial taxonomies from ACM, IEEE, GoogleCode and SourceForge. A listing of the starting categories is provided in Table 1.

Step 2: Conduct multiple tool-supported individual brainstorming sessions: We sent targeted requests to personal contacts in a wide variety of software organizations such as Canadian universities, IT departments within the Canadian federal government, and software consulting companies.

Table 1: Seed categories for the taxonomy-building application.

Audio	Office/Business
Clustering	Operating Systems
Communications	Plugin
Database	Printing
Desktop	Programming
Development Tools	Religion
Distributions and Standards	Scientific / Engineering
Education	Security
Enterprise	Sociology
Financial	Storage
Games/Entertainment	Student
Graphics	System and Network
Hardware	Utilities
Information resources	Terminals
Linux	Text Editors
Management and Business	Utility
Miscellaneous	Voice Over IP (VoIP)
Multimedia / Publishing	Web / Internet
Networking / Telecom	Windows
	XML

In total, we collected data from 78 sources, of which 53 were volunteer participants. The software practitioners we contacted provided us with 11 more-or-less complete taxonomies. An additional 42 participants provided a small segment of a software taxonomy; either because they chose not to continue with the research, or they restricted their opinions to their area of expertise. The participants had a range of education from a community college diploma to a master's degree; they all lived in Canada or the USA. An additional 25 full or partial taxonomies were compiled from existing websites such as GoogleCode, and SourceForge.

Each participant used a tool described in the next section to brainstorm a complete software taxonomy starting with a random sample of four to seven categories from Table 1

The volunteers were asked to organize software application categories into high-level software groupings. They were able to add new categories and groupings based on personal experiences. The participants did not need to concern themselves with details such as the order of subcategories within a super-category.

Step 3: Merge the brainstorm results to form a full first draft. Next, we merged all of the expert defined taxonomies from Step 2 into a draft tax-

onomy. The initial merged structure incorporated the results from Step 1 and 2 above.

Step 4: Refine the taxonomy by applying systematic criteria for subdivision. Our next objective was to improve the rationale for each division and fine tune the overall structure. With a draft taxonomy in hand, we were able to define the design criteria from which additional taxonomy elements could be systematically added. The criteria we arrived at are described in Section 3.

Step 5: Review and edit the result iteratively. To refine our candidate taxonomy we conducted an iterative review process with six software practitioners to arrive at the taxonomy presented in Section 3. The iterative review process meant that changes resulting from each review were incorporated immediately, so were visible in later reviews.

The tools used to enable both the original taxonomy creation and the iterative review are described in the next section.

2.2 Software Taxonomy Tools

We built an online tool called *SoftwareTaxonomy* [10] to facilitate the creation, manipulation, and analysis of our taxonomy of software application types.

To encourage proper use of the application, we created training videos. Our first video was to promote the importance of our work and the second video was a tutorial to describe by example how to use the software. The videos [11],[12] are hosted on YouTube and they are embedded within *SoftwareTaxonomy*.

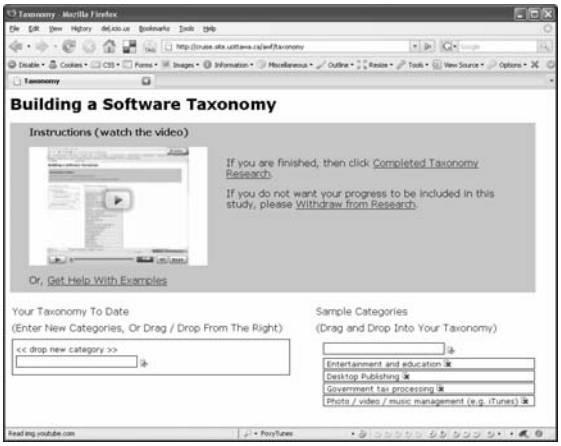

Figure 1: Entering a taxonomy from scratch

As mentioned in Step 2 in the previous section, our method involved having participants brainstorm a complete software taxonomies starting with just a few seed categories. A screenshot of the taxonomy application is presented in Figure 1. The participants could drag and drop sample categories, add new categories and re-arrange the categories into the preferred order.

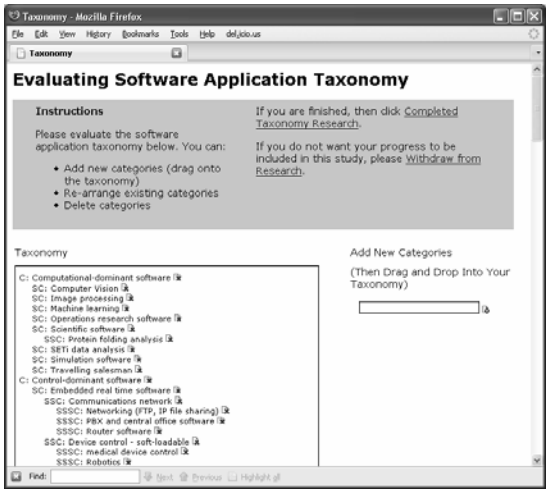

Figure 2: Peer-Review tool for the Software Taxonomy

In Step 5, also described in the previous section, participants were able to peer-review the application taxonomy so that it could reach its final form.

Figure 3: Reviewing the software taxonomy in MS Word

For this process, some users opted to communicate via email, so a Microsoft Word version of the taxonomy was therefore made available.

As feedback was provided, both taxonomies (MS Word and online in *SoftwareTaxonomy*) were updated to allow subsequent peer reviews to be based on the most up-to-date version – allowing a distributed collaborative exchange among software practitioners.

Figure 2 shows the online version and Figure 3 shows the offline Microsoft Word version for this peer-review and editing process.

3 Results

In this section we present the proposed taxonomy resulting from the process described in the last section. The complete taxonomy is presented in Section 3.4, and is available online [13].

We used a systematic coding scheme for labelling the categories. The top level uses a letter from the set {A, B, C, D}. We expect this level to be very stable in further evolution of the taxonomy. The second level uses 2 or 3 lower-case letters representing a mnemonic code for the category, so for example 'A.bus' would be the 'Business-oriented software' subcategory, in top-level category A. A summary of the taxonomy, showing just these first two levels is shown in the next section

The third and fourth levels use a number and a letter respectively. We expect considerable expansion and evolution in these levels.

3.1 Top Levels of the Proposed Taxonomy

The top two levels of the proposed taxonomy are presented in Figure 4.

At first glance, some of the categories in these levels appear similar. Below we provide more details about certain categories and in particular explain the differences between similar ones.

The A.inf category includes software applications that involve the display and dissemination of information as well as *front-end* maintenance of this information. Typical applications in this category include digital-maps, news, weather, and booking system. In contrast, D.im software involves a much higher degree of data manipulation to enable decision making such as inventory systems, sales forecasting and search engine processing.

Software in category B.ut includes tools, diagnostics, and maintenance software to enhance the capabilities of the underlying computer / system; examples include process viewers, logging tools, and backup utilities.

C.hw software controls the direct manipulation of hardware; for example, software within your printer. In contrast, C.em software is software embedded within hardware that may provide additional services, not just the control of the hardware – for example, a speech coder for a cell phone. Finally C.rt software has specific real-time needs usually associated with safety-critical software such as a pacemaker.

D.art refers to the highly specialized software that deals with the manipulation of artistic media such as audio and video including video processing, audio mastering and graphic manipulation. A.con, also contains creative software (among other things), but it would be consumer oriented.

A Data-dominant software
- **A.con** Consumer-oriented software
- **A.bus** Business-oriented software
- **A.des** Design and engineering software
- **A.inf** Information display and transaction entry

B. Systems software
- **B.os** Operating systems
- **B.net** Networking / Communications
- **B.dev** Device / Peripheral drivers
- **B.ut** Support utilities
- **B.mid** Middleware and system components
- **B.bp** Software Backplanes (e.g. Eclipse)
- **B.svr.** Servers
- **B.mal** Malware

C. Control-dominant software
- **C.hw.** Hardware control
- **C.em.** Embedded software
- **C.rt.** Real time control software
- **C.pc.** Process control software (i.e. air traffic control, industrial process, nuclear plants)

D. Computation-dominant software
- **D.or.** Operations research
- **D.im.** Information management and manipulation
- **D.art.** Artistic creativity
- **D.sci** Scientific software
- **D.ai** Artificial intelligence

Figure 4: Software Taxonomy Level 1 and 2

These four top-level categorizations are fairly well balanced with two categories featuring four sub-categorizations, one with five, and the largest with eight sub-categorizations.

3.2 Criteria for Dividing Categories at Each level

An analysis of the merged taxonomy from Step 3 resulted in the creation of rules (as outlined in Step 4) to make the division and ordering of successive layers of subcategories more systematic. The classification regime is summarized in Table 2.

Table 2: Criteria for organizing the taxonomy

Level	Parent	Criteria for subdividing
1	Root	Data Dominant to Computation Dominant
2	A. Data-Dominant Systems	Target user audience (consumer, designer, and information-seeker)
2	B. Systems software	Level of abstraction (low to high-level)
2	C. Control-dominant software	Scale of control (small to large)
2	D. Computation-dominant software	Conceptual dimension (practical to theoretical)
3	All	Application domain (e.g. communication, entertainment, education)
4	All	Features / functions (e.g. web browsers, email, financial analysis)

The four first-level categories divide software based on whether A data management, B system services, C active control or D computation is more important. For each of these top-level categories, different criteria are then used to divide the hierarchy at the second level.

The categorizations are defined as follows:

- **Data-dominant systems**: what is the specialization of the target user audience from general purpose consumer-oriented, to more specific business software to engineering / technology users.

- **Systems software**: what level of abstraction are you achieving from low-level hardware abstraction of operating systems to high-level software backplanes like Eclipse).

- **Control-dominant software**: how much are you controlling the system from small embedded systems to extremely large operations like an airport).

- **Computation-dominant software**: how conceptual is your problem from practical software like the SETi program to theoretical areas like artificial intelligence.

Level 1 provides a high-level classification of software applications divided into four groups loosely based on the dominating concern dealt with by the software. Level 2 is a finer-grained classification that is specific to the high-level group.

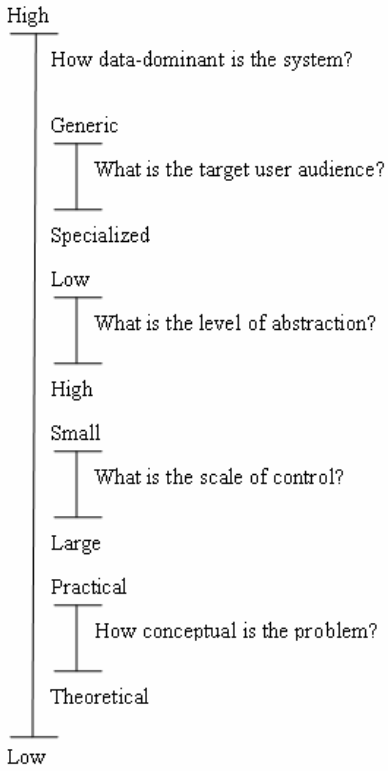

Figure 5: Visualizing taxonomy design criteria

The third-level classification is the same for all paths of the taxonomy and reflects the more specific application domain – which have been amassed from various sources including the ACM, Google and SourceForge.

The fourth-level classification is also the same for all paths. This level reflects the predominant functions / features of an application such as browsing, emailing, or financial analysis.

Figure 5 shows visually how the first two levels of the software taxonomy are classified – including the qualitative scale used (e.g. high to low).

For example, the Level-2 subdivision criteria for data-dominant systems is by their target audience (from generic to specialized); whereas, the Level-2 subdivision for systems software considers the level of abstraction of the software (from low to high). These different categorization scales resulted from the analysis outlined in Section 2. In particular, it was observed that each sub-category within a top-level {A, B, C, D} is organized based on different concerns, resulting in different classification rules to organize a particular subcategory.

3.3 Statistics Regarding the Taxonomy

The full taxonomy, presented in the next section, is not a balanced tree. The 'A. Data-dominant systems' and the 'B. Systems software' top-level categories contain more entries and include more levels of categorization then do 'C. Control-dominant software' and 'D. Computation-dominant software'. The imbalance observed above is partly due to the nature of software development. The software participants and online resources (including open-source hosting sites like GoogleCode and SourceForge) that helped shape and validate the proposed taxonomy primarily focus on data-dominant systems.

The imbalance of the taxonomy does not necessarily mean that the research methods used to build the taxonomy are flawed, or that the taxonomy itself is not useful. As new software application types are discovered, researchers can use the categorization technique described in this paper to help properly place those application types. And, by being aware of the unbalanced taxonomy, researchers can avoid making overly broad statements within the less populated areas of the structure.

The full taxonomy, as presented in the next section, has 191 categories. As seen in Table 3, each category in the first two levels of the taxonomy has around five sub-categories. Only 11 of the 81 third level categories are further subcategorized (each with approximately eight fourth level subcategories).

Table 3: Software taxonomy distribution

		Number of Children		
Level	**N**	**mean**	**st. dev**	**N without children**
1: Data / Computation	4	5.3	1.9	0
2: Scale	21	5.4	3.7	6
3: Domain	81	7.7	3.9	70
4: Function	85			
	191			

Mean and standard deviation do not include categories that do not have children

3.4 Proposed Taxonomy

The full proposed taxonomy is presented below. It is also available online at [13]. Please note most of the taxonomy is categorized to three levels of detail; most of the fourth level of detail is present in the Data-dominant systems.

A Data-dominant software
- **A.con** Consumer-oriented software
  - **1** Communication and information
    - **a** Voice
    - **b** Text / Chat
    - **c**. Email
    - **d** Web browsers
    - **e**. Personal Information management (calendaring, address book)
    - **f**. File / document sharing (i.e. FTP)
  - **2**. Productivity and creativity
    - **a**. Text Editors
    - **b**. Word processing
    - **c**. Spreadsheets / Calculators
    - **d**. Presentation (e.g. PowerPoint)
    - **e**. Desktop Publishing
    - **f**. Personal time management
  - **3.** Entertainment and education
    - **a.** Learning and Reference
    - **b.** Training / Courseware
    - **c.** E-Books
    - **d.** Photo / video / music management (e.g. iLife)
    - **e.** Media players (e.g. music, graphics, photo etc)

f. Movies / Animations / Audio (as software as opposed to pure data or players)
g. Games

4. Personal management
a. Genealogy
b. Personal finance / budgeting
c. Personal will / legal assistance software
d. Tax preparation / planning
e. Monitor / tracking software (e.g. training / health)

A.bus Business-oriented software
1. Strategic and operations analysis
a. Statistical / risk analysis
b. Financial analysis and enterprise resource planning
c. Legal analysis / assistance
d. Domain specific database searching (e.g. patent, trademark off-line searches)
e. Workforce Management (e.g. time and attendance software)
f. Payroll and human resources management
g. Project management / workflow
h. Procurement
i. Scheduling and logistics
j Medical diagnosis

2. Corporate management
a. Real-estate management
b. Restaurant management / reservations / meal planning
c. Sales management
d. Portfolio management
e. Hospital management
f. Facilities management

3. Information management and decision support systems
a. Data warehousing
b. Expert systems
c. Management information systems
d. Knowledge / records / information management system
e. Product support, help desk system
f. Health information / online medical records
g. Geographical information systems
h. Data mining / business intelligence management systems

4. Transaction processing
a. Accounting
b. Payroll
c. Inventory management
d. Order management and Billing support
e. Bank transaction processing
f. Government tax processing
g. Transportation reservation (airline, train, bus)
h. University / school timetabling and registration
i. Reward program backends
j. Bulk data analysis

A.des Design and engineering software
1. Development environment
a. Implementation tools (i.e. editors, refactoring, code-assist, etc)
b. Version control / configuration management
c. Process support tools (i.e. debuggers, context-sensitive help, etc)
d. Development environment plugins
e. Software development simulation / prototyping

2. Compilers / interpreters / disassemblers
3. Automatic Code Generation
4. Database, Development, Reporting
5. CAD / CAE tools
6. Modeling / CASE tools
7. Drafting / Architecture tools
8. CAM (Computer-aided manufacturing) tools
9. Engineering analysis
10. Benchmarks
11. Software testing tools

A.inf Information display and transaction entry
1. Information resources
a. Languages (Spoken / Written)
b. Libraries
c. Maps & Travelling
d. Contact information software (e.g. phone book)

2. Standalone applications for displaying information

3. Web applications / services
a. Search engines
b. News and media
c. User-generated content (e.g. Blogger, Wordpress, message boards)
d. Social networking (e.g. Facebook)
e. Weather
f. Job search
g. Maps and navigation
h. Online dictionaries and encyclopaedias
i. Online booking (travel, entertainment)
j. Online productivity software
k. Website content management
l Web Authoring / Publishing Software
m. Public library systems
n. Enrolment / registration in education
o. E-Commerce
p. E-finance (banking, investing, money transfer)
q. E-Government (passports, birth certificates, vehicle registrations, gun registration, unemployment insurance apps. etc.)
r. E-Democracy (voting online)

B. Systems software
B.os Operating systems
1. Accessibility
2. Administrator software and tools

3. Emulation / emulators
4. Games console operating systems
5. Virtual machines
6. Kernels / distributions (e.g. Linux, Mac, Windows, Palm, QnX)

B.net Networking / Communications
B.dev Device / Peripheral drivers
B.ut Support utilities
1. Anti-virus / anti-spyware / firewalls
2. Authentication tools
3. Backup, Recovery, Storage
4. Compression / decompression
5. Data format conversion
6. Diagnostic / process viewer / activity monitor
7. Disk maintenance
8. Encryption / decryption
9. Failure diagnosis
10. Logging and log analysis
11. Screen capture
12. Network traffic monitor
13. Security
14. Software installer / uninstaller
15. Tunnelled communication network client (e.g. VPN)
16. Wireless Utilities

B.mid Middleware and system components
1. Database servers
2. Graphics packages / rendering engines
3. Interoperability infrastructures
4. UI support software
5. Virtual machines
6. Windowing system servers

B.bp Software Backplanes (e.g. Eclipse)
B.svr. Servers
1. Email servers
2 IM servers
3. Load balancers
4. Proxy servers
5. Web / FTP / Content servers
6. Other daemon processes

B.mal Malware
1. Key loggers
2. Spyware
3. Viruses / Trojans

C. Control-dominant software
C.hw. Hardware control
1. Firmware (e.g. software in printers, DVDs, watches, fridges, etc)
2. Device control - soft-loadable (e.g. robotics, medical, e-voting, etc)

C.em. Embedded software
C.rt. Real time control software
C.pc. Process control software (i.e. air traffic control, industrial process, nuclear plants)

D. Computation-dominant software
D.or. Operations research
1. Computer science hard problems (i.e. Travelling salesman)
2. Simulation software

D.im. Information management and manipulation
1. Inventory control
2. Sales forecasting
3. Budget generation / management
4. Search engine processing

D.art. Artistic creativity
1. Photo, drawing, graphics editing / manipulation
2. Audio & recording, mastering
3. Music composition (audio editing / synthesis)
4. Movie creation (film / movie production)
5. Video processing (editing, surveillance, recognition)

D.sci Scientific software
1. Idle-time data analysis (e.g. SETi)
2. Simulation software
3. Signal analysis software
4. Image processing
5. Computer Vision

D.ai Artificial intelligence
1. Agents
2. Machine learning
3. Virtual Reality
4. Robotics / Cybernetics

3.5 Comparing Our Taxonomy

If we compare the ACM taxonomy for software to our software taxonomy, it can be seen that we included most of the Section J subcategories but omitted the following:

- Freeware/shareware
- Psychology
- Sociology
- Marketing
- Military

The Freeware/shareware category cannot be classified in our taxonomy – as it represents a mode of development common to many application domains as opposed to an application domain itself. The remaining four categories have been omitted as they are occupation-oriented, and would have aspects that could fall in many of our

categories. For example, military software includes embedded software, control software, and financial software – all of which appear more appropriately elsewhere in our taxonomy.

As well, the following ACM entries are recategorized under scientific software in our taxonomy, whereas in the ACM taxonomy they are categorized mostly Section J.2 Physical Sciences and Engineering as well as J.3 Life and Medical Sciences.

- Chemistry (J.2)
- Earth and atmospheric sciences (J.2)
- Biology and genetics (J.3)
- Physics (J.2)
- Aerospace (J.2)
- Archaeology (J.2)
- Astronomy (J.2)

Again as opposed to categorizing by field, our approach was more closely tied to the type of scientific software in use such as Idle-time data analysis (e.g. SETi) and Simulation software.

The following entries were omitted from Google's application list:

- Mobile Computing (724)
- Home (9)
- Marketing (449)

These categories are too ambiguous or broad in scope to be considered as leaf categories in our taxonomy, and do not align with our classification criteria. If individual software application types are considered, each would find the appropriate category in our taxonomy. For example, the Mobile Computing and Home categories involve networking, productivity software, embedded devices, etc., whereas the Marketing category could be classified under Data-dominant applications, and the particular sub-category depends on the goal of the application (e.g. financial analysis versus multi-media players).

Overall, both the ACM taxonomy and GoogleCode lists provided a great reference point to help define the guiding criteria, as well as populate the taxonomy with appropriate application types.

4 Concrete Examples

We have identified concrete examples and situations where our proposed application taxonomy could be useful. For academic research the primary benefit is to help label either (a) under what circumstances research was performed, or (b) where software processes / tools / methodologies seem to fit best. For corporate environments the primary benefit is to be able to know whether the evidence an engineer intends to rely on is applicable to their particular application domains.

4.1 Academic Research

In an academic environment, the taxonomy can be applied in the following scenarios

- Surveys and questionnaires should use the first and/or second levels of the taxonomy to gather demographic information about the participants. This information would allow for more standard sub-sampling among different studies.
- Experimental or other empirical results should indicate which application domains were used in the study.
- Retrospective analysis should be performed on existing research that gathered *application type* information. Existing research that may have recorded the types of applications involved in the study can now be grouped at a more abstract categorization (i.e. levels one and two). Similar to the first scenario, researchers now have a more structured approach to combining results to uncover similarities amongst sub-samples.

4.2 Academic Education

For the purposes of education, the taxonomy will enable educators to either ensure depth of study into a particular taxonomy categorization, or a breadth of coverage to promote exposure to all (or at least most) of the varieties of software. In both cases, the students will benefit from an increased awareness of how software types are interrelated or differ. In particular:

- Curriculum designers can review course selection to promote a certain niche (e.g. software gaming) or ensure a broad coverage (e.g. general software engineering)
- Academics can create case studies, textbooks, and problem sets that indicate the application types to which they are relevant.

In additional to the above, and somewhat related to the previous section of academic re-

search, professors in the field of software education can now better evaluate course processes and tools against the type of software being built to better understand under what circumstances do certain software tools flourish and/or fail.

4.3 Software Practitioners

Practitioners are more likely to adopt techniques that have been demonstrated on similar systems. But, to what level of detail is it required to define *similar*.

Taxonomies by nature presume that similar elements are close to one another in the hierarchy and dissimilar elements are far away. Although our work cannot answer to what extent similarity exists, individual researchers as described in the previous section can now provide quantitative analysis demonstrating the applicability of a certain result within a categorization, and amongst sub-categories.

Based on evidence in the literature that relates various tools, techniques and methods to specific application categories, the software taxonomy will help guide software practitioners when making decisions about items such as the following:

- Software methodologies.
- Design patterns and component frameworks.
- Algorithms.
- Software engineering tools such as IDEs,
- Data and exchange formats such as XML, YAML, and JSON.
- Architectures and architectural styles.
- Testing techniques including those for unit, integration, system, user acceptance and smoke testing.

In general, software practitioners will be better able to compare what has worked versus what has not worked for a particular type of software development.

5 Threats to Validity

The main threats to validity of our work are summarized below. We have also outlined the steps we have taken to help mitigate these threats.

Sample bias (non randomized). The primary threat of this work is that our data may not reflect the general software engineering community as we may have only been able to attract participation from a certain category of software practitioners with potentially non representative views of the desired structure for a software taxonomy. To reduce this threat and help ensure that our sample was based on a representative collection of software practitioners we approached both open and closed forums for participation. We also incorporated well known software lists from organizations like Google and ACM.

Researcher bias. Ultimately it was at the discretion of the researchers how the software taxonomy was structured. To minimize this threat, we concluded our research with an iterative review of the taxonomy and all comments and changes were carefully considered.

These threats will minimize in time as the taxonomy continues to be used and reviewed by external researchers and practitioners. As this occurs we will be more confident about the completeness of the taxonomy.

6 Conclusions and Future Work

In this paper we have presented a proposal for a taxonomy of software types. This can be used to better focus and classify research. It can also be used by practitioners to better assess whether research results are relevant.

We would like to see a standards organization adopt a descendent of our proposal, but we fully recognize that iteration and evolution will be needed before that happens. In the meantime, we would like to see researchers and practitioners, who have no current alternative taxonomy, use-our taxonomy on an experimental basis. They can use it to classify empirical results, to tag libraries, plugins, patterns and algorithms, and as the basis for reference models, frameworks and case studies.

Researchers that are interested in enhancing our taxonomy could use the taxonomy software we created to develop it further.

The end result would be that the field of software engineering would gain more solid evidence that a particular methodology, development style or testing strategy is broadly or narrowly applicable.

Acknowledgements

The author would like to thank our industrial partners at IBM CAS Ottawa, and all those who contributed to our taxonomy.

About the Authors

Andrew Forward is a PhD student at the University of Ottawa. He has industrial experience with Deloitte Consulting and IBM.

Timothy C. Lethbridge is a professor of computer science and software engineering at the University of Ottawa. He conducts research that focuses on complexity reduction in software engineering. His areas of interest include user interfaces, software design and software engineering education. He is author of a McGraw-Hill textbook on software engineering.

References

[1] Tzitzikas Y, Spyratos N, and Constantopoulos P. Mediators over taxonomy-based information sources. The VLDB Journal 2005; 14: 112-136.

[2] Landwehr CE, Bull AR, McDermott JP, and Choi WS. A taxonomy of computer program security flaws. ACM Comput Surv 1994; 26: 211-254.

[3] Vessey, I. 1997. Problems versus solutions: the role of the application domain in software. *Seventh Workshop on Empirical Studies of Programmers* (Alexandria, Virginia, United States). S. Wiedenbeck and J. Scholtz, Eds. ESP '97. ACM, New York, NY, 233-240.

[4] I. Vessey, Cognitive fit: a theory-based analysis of the graphs versus tables literature, Decision Sciences 22 (2), 1991, pp. 219-240.

[5] Glass RL, Vessey I. Toward a taxonomy of software application domains: History. J Syst Softw 1992; 17: 189-199.

[6] Glass RL, Vessey I. Contemporary application-domain taxonomies. IEEE Softw 1995; 12: 63-76.

[7] Jackson, M., "Problem frames and software engineering", Journal of Information and Software Technology, 2005, vol 47, no 14, Nov 2005, pp.903–912.

[8] Jackson, M., "Problem Frames: Analysing, Structuring Software Development Problems". Addison-Wesley, 2001.

[9] Glass RL. Matching methodology to problem domain. Commun ACM 2004; 47: 19-21.

[10] Forward A. Study: Building a software taxonomy. Retrieved 12/14, 2007, from http://cruise.site.uottawa.ca/awf/ .

[11] Forward A. Why build a software taxonomy? Retrieved 12/14, 2007, from http://www.youtube.com/watch?v=iqlnWIFZ5BM .

[12] Forward A. Tutorial - How to create a software taxonomy. Retrieved 12/14, 2007, from http://www.youtube.com/watch?v=Ut2O8xzeIMI .

[13] Forward A. Latest software taxonomy. Retrieved 12/14, 2007, from http://www.site.uottawa.ca/~tcl/gradtheses/aforwardphd/Software%20Taxonomy.doc

[14] Forward, A. Modeling survey data and software application taxonomy data available at http://site.uottawa.ca/~tcl/gradtheses/aforwardphd/.